VOLUME 2 ISSUE 3 $1.50 NOVEMBER 1985

SUPER 99 MONTHLY

Millers Graphics has announced the upcoming release of a new software package, DisKassembler™. Written by Tom Freeman, DisKassembler™ creates directly assemblable source files from 99/4A Assembly Language object code that is in either Display Fixed 80 or memory image format (such as game files). In addition, it will disassemble console memory and all valid DSR's. Program output is to disk or any printer. Object files may be from floppy disk, hard disk or RAM disk in CorComp, MYARC or TI disk controller formats. The program is for anyone interested in how programs were constructed and in learning new programming techniques. Carrying a suggested price of $19.95 (plus shipping and handling), the package will include complete and useful documentation (the hallmark of all MG products).

The first shipments of GRAM Kracker™, Millers Graphics' incredible new hardware device, will be released on December 16 and 17. Due to quality control procedures that ensure that all customers will receive the product without jumper modifications, the shipment dates are behind original projections, which has prompted Millers Graphics to provide UPS Blue Label shipping at no extra charge to ensure arrival by Christmas. As the 3 optional RAM chips for GRAM Kracker™ have been reported to be difficult to find in some regions, MG now offers the chips at $4.50 each, with C.O.D. ($1.90) being available for U.S. customers (other countries, prepaid). Installation is provided only for orders initiated with the optional chips specified (total price $184.95 plus shipping and handling).

FORTH

Strings, Part 1

by Warren Agee

STANDARD: 1A 2EA 4B 5A 6B 7B 9B

PREFACE:

With this tutorial (and more to come!), I humbly submit what I have learned by programming in the FORTH language. One reason I decided to put down into words the knowledge I have acquired is to share my experiences, frustrations and triumphs while hacking away with FORTH. But, on a more personal level, I give these tutorials to the TI world as a token of appreciation for everything I have gained from knowing such people as Ronald Albright, Barry Traver, and Howie Rosenberg, just to name a few, as well as the whole gang on the TI FORUM. These and many others have given unselfishly to both me and the TI community as a whole, and I am proud to be part of a community that refuses to die. Now, on with the programming, FORTHwith! <ugh!>

-->

STRINGING ALONG IN FORTH

Of all the peculiarities the beginner confronts in FORTH, string handling is a major obstacle. Nothing is more frustrating than to sit down and have no idea how to write something like A$="1234"::A=VAL(A$). No advanced string-handling routines come with the TI FORTH systems disk. So, it is up to the programmer to invent his own. Hopefully, this article will make it much easier to write a FORTH program that involves any string mainpulation at all.

THE BASICS

Before jumping into the new string words, let's first take a look at how a string sits in memory. This knowledge is imperative in order to fully exploit the power of FORTH. Think of a string as a numeric array; each character in the string represents a number, or byte. The string HOME COMPUTER would look like this:

```
---------------------------
|H|O|M|E| |C|O|M|P|U|T|E|R|
---------------------------
```

The first "box" represents the address in memory where this string starts. Determining the location of this address is what we will discuss next.

There are many ways to store strings; we could save them in VDP RAM, or in the disk buffers. In this article, we will investigate storing strings directly in the dictionary. A string variable is no more than a numeric variable stretched out. In fact, unlike BASIC, there is only one type of variable in FORTH. The only thing that differs is the size. First use the word VARIABLE to create a variable. But when you create it, let's say O VARIABLE TEST, only two bytes are alloted for storage. This is fine for single numbers; but for strings, we can use ALLOT to specify the length of the variable. For instance, O VARIABLE TEST 8 ALLOT will create a variable with a length of ten bytes. This gives us room for a string with a maximum length of 10 characters. If the above is exectuted, the variable will look like this in memory:

```
---------------------
| | | | | | | | | | |
---------------------
|
|
addr of TEST
```

Once the string is created in the dictionary, there may be garbage in the variable. Here we can use BLANKS to clean it out: TEST 10 BLANKS. This will fill ten bytes of memory, starting at TEST, with blanks (ASCII 32).

Now that space has been reserved for the string, there are basically two ways to store the string. If the contents of the variable is not going to change, then the word !" can be used. All this word requires is an address on the stack. So, to store STRINGS in the variable TEST defined above, the sequence TEXT !" STRINGS" will do the trick. If you wish the user to input the string, the word EXPECT is available, which is similar to BASIC's INPUT statement; it awaits an entry from the keyboard. EXPECT requires both an address and the maximum length of the string on the stack. Using TEST 7 EXPECT will achieve the same results as TEST !" STRINGS" . The variable will now look like this:

```
---------------------
|S|T|R|I|N|G|S| | | |
---------------------
```

This presents our first problem. Since the contents of TEST is not expected to change, the length of the string can be assumed to always be 7. However, if the length will vary, we must keep track of it. EXPECT does not do this for us. Sure, it requires a length on the stack, but it does not incorporate this value into the string. Not to worry. This brings us to our first new word, ACCEPT, which replaces EXPECT. The only difference is that ACCEPT stores the actual length of the string entered into the byte preceding the string. This is often called the count byte. If we use ACCEPT in the example above, our string would now look like this:

```
---------------------
|7|S|T|R|I|N|G|S| | |
---------------------
 |
 |
 addr of TEST
```

-->

As you can see, the first letter of the string, the "S", no longer sits at TEST; the whole string has moved over one byte to make room for the count. Now, to print this string is a trivial matter of using TEST COUNT TYPE. TEST supplies the addr of the complete string. COUNT takes that address, calculates the address of the actual string (TEST+1), and finally supplies the length of the string. Everything is ready for TYPE. To summarize what we have done so far, consider the following example:

```
0 VARIABLE COOKIE 18 ALLOT (reserves 20 bytes)
COOKIE 20 BLANKS
COOKIE 20 ACCEPT _CHOCOLATE CHIP_
COOKIE COUNT TYPE
```

Note: any words that appear between underscore characters (_) are to be typed in as a response to the ACCEPT word.

MOVING AROUND

Up till now, I have discussed performing basic functions on strings which reside directly in the dictionary. This is not always the ideal situation. A much better way is to store the string in a temporary spot, do what needs to be done, then move it back into the dictionary. This temporary spot is called PAD. Typing in PAD just leaves an address on the stack, just as TEST does. Typically, instead of typing in TEST 10 ACCEPT, you would type PAD 10 ACCEPT. Once any processing is done, the word CMOVE can move the bugger back to where it belongs. Here arises our second problem. CMOVE moves a specified quantity of bytes from low memory to high memory. But what if you want to go the other way around? Well, define a new word, of course! The new word will be <CMOVE, which is included in some versions of FORTH. But wait--isn't it rather a hassle having to remember which word to use? Of course it is! Remember, FORTH is extensible, and we can make it as user-friendly as we like! The next new word will be CMOVE$, which decides which way the string is moving, and does the moving for you.

Here is an example of using CMOVE$ and PAD:

```
0 VARIABLE DRESSER 8 ALLOT
DRESSER 10 BLANKS
PAD 10 ACCEPT  _SOCKS_
.
. (string processing done here)
.
PAD COUNT                    (get addr and length)
1+ SWAP 1- SWAP              (PAD-1 CNT+1)
DRESSER SWAP                 (PAD-1 DRESSER CNT+1)
CMOVE$
DRESSER COUNT TYPE
```

Everything should make sense until you get to the 1+ SWAP 1- SWAP. The reasoning is a little hard to grasp at first: we want to move SOCKS from PAD to DRESSER. We also want to maintain that ever-important count byte. But when we use PAD COUNT, we only have the addr and length of the string itself, not including the count. So we compensate. Add 1 to the count (because we want to move the count byte along with the string), then subtract one from the address. COUNT adds 1 to the address, so we have to correct this to catch the count. Once these two numbers have been corrected to catch the count byte, shift things around to get everything ready for CMOVE$. To better illustrate this, here is a diagram of PAD:

```
 -------------------
 |5|S|O|C|K|S| | | | |     (Contents of PAD)
 -------------------
  | |
  | PAD+1 (This is where you are using PAD COUNT)
  |
 PAD (This is where you are using PAD COUNT 1+ SWAP 1- SWAP)
```

If you can understand the principle of the count byte, and how to keep the count byte tacked on to the string when moved, then a major obstacle in writing in FORTH has been removed. Next time, I will discuss string arrays. Until then, experiment, and Keep On FORTHin'!

-->

SUMMARY OF RESIDENT WORDS -

```
VARIABLE (n--)          Create a variable.
ALLOT    (n--)          Reserves n bytes in the dictionary.
BLANKS   (addr n--)     Fills n bytes with blanks.
EXPECT   (addr n--)     Waits for input; stores string at addr.
COUNT    (addr--)       Returns addr and count of a string.
CMOVE    (adr1 adr2 n)Moves n bytes from adr1 to adr2, from low to
                        high memory.
PAD      (--adr)        Temporary storage place for strings.
```

NEW WORDS

```
: PICK  ( n1 -- n2)

  2 * SP@ + @ ;

( Copies n1th number to top of stack)
******

: LEN   (addr -- n)

  255 0 ( string max=255 characters)
  DO
     DUP I + C@
     0= IF        ( looks for null)
        I LEAVE   ( I=length of string)
     ENDIF
  LOOP
  SWAP DROP ;

( Returns the length of a string at addr.)
******

: ACCEPT  ( addr n -- )

  OVER 1+ DUP ROT    ( adr+1 )
  EXPECT
  LEN                ( length of string)
  SWAP C! ;          ( store count byte at addr )

( Waits for input; stores count at addr and string
  starting)
( at adr+1.)
******

: <CMOVE  ( adr1 adr2 n)

  DUP ROT + SWAP ROT
  1-DUP ROT +
  DO
    1- I C@ OVER C! -1
  +LOOP
  DROP ;


( Moves n bytes from adr1 to adr2, from high to low memory.)
******

: CMOVE$  (adr1 adr2 n)

OVER 4 PICK >
IF <CMOVE
ELSE CMOVE
ENDIF ;

( Moves n bytes from adr1 to adr2; automatically decides on)
( direction.)
```

-->

ASSEMBLY

STANDARD: 1A 2XB EA TW 3B 4B 5A 6B 7B 9B 10B

```
*****************************************************************************
*   TI-WRITER SCREEN DUMP  inspired by May, 1985 Super 99 Monthly           *
*                                                                           *
*   The following Source code, when assembled and combined with the XB      *
*   calling routine and Subprogram will create a DISPLAY/VARIABLE 80 file   *
*   that will print a screen image from the TI-WRITER FORMATTER.            *
*                                                                           *
*   The program will work with any EPSON compatible printer.                *
*                                                                           *
*   Insert the following line in your XB program where you want the dump to *
*   occur:                                                                  *
*                                                                           *
*       CALL TIW_DUMP(DE,F$,BL,EL,T):: STOP                                 *
*                                                                           *
*           Where   DE= Density  (1 or 2)                                   *
*                   F$= Filename that you want the dump stored under        *
*                       For example:   DSK1.PICTURE                         *
*                   BL= Beginning line of the screen that you want saved    *
*                   EL= Ending line of the screen that you want saved       *
*                   T = Tab value   Note: Tab of 20 centers picture         *
*                                                                           *
*   Type in and save the following sub program in merged format.  Merge it  *
*   into the program that contains the graphics that you want dumped.       *
*                                                                           *
*       25000 SUB TIW_DUMP(DE,F$,BL,EL,T)                                   *
*       25010 ON ERROR 25080                                                *
*       25020 IF (T<0)+(T>40)+(BL>EL)+(BL<1)+(BL>24)+(EL<1)+(EL>24)         *
*       THEN GOSUB 25080                                                    *
*       25030 IF DE<>2 THEN DE$="DE1" ELSE DE$="DE2"                        *
*       25040 CALL INIT :: CALL LOAD("DSK1.TIWDUMP-O"):: CALL LINK(DE$,     *
*       F$,BL,EL,T)                                                         *
*       25045 ! LINES 25050 to 25070 MAY BE DELETED IF DESIRED              *
*       25050 OPEN #1:F$,DISPLAY,VARIABLE 80,APPEND                         *
*       25060 PRINT #1:CHR$(27)&CHR$(64):".PL 1" ! 27-64 RESETS PRINTER,    *
*       .PL 1 WILL STOP UNWANTED FORM FEED                                  *
*       25070 CLOSE #1                                                      *
*       25075 SUBEXIT                                                       *
*       25080 PRINT "BAD PARAMETER" :: STOP :: RETURN                       *
*       25090 SUBEND                                                        *
*                                                                           *
*****************************************************************************
*                                                                           *
*   by Joseph H. Spiegel                                                    *
*      SOURCE ID: TI6240     COMPUSERVE ID  72426,3432                      *
*                                                                           *
*****************************************************************************
       DEF   DE1,DE2
VSBW   EQU   >2020
VMBW   EQU   >2024
VSBR   EQU   >2028
VMBR   EQU   >202C
STRREF EQU   >2014
NUMREF EQU   >200C
FAC    EQU   >834A
       AORG  >2700
DE1    MOV   R11,@SAVE      SAVE RETURN ADDRESS
       LWPI  MYREGS
       CLR   R14            RESET FLAG -> SINGLE DENSITY
       JMP   MAIN
DE2    MOV   R11,@SAVE      SAVE RETURN ADDRESS
       LWPI  MYREGS
       SETO  R14            SET FLAG -> DOUBLE DENSITY
*****************************************************************
*      GET START AND END LINES AND TAB INFO                     *
*****************************************************************
MAIN   LI    R4,STARTL      POINT TO LOCATION TO HOLD START ADDRESS
       LI    R1,2           START LINE IS SECOND VALUE FROM XB
GLINE  CLR   R0
       BLWP  @NUMREF        GET VALUE PASSED FROM XB
       MOV   @FAC,R5        MOVE VALUE FROM FAC TO R5
       ANDI  R5,>00FF       VALUE IS IN LOWER BYTE
       DEC   R5             LINE 1 STARTS AT >0000V
       SLA   R5,5           X32 BYTES PER LINE
```

-->

```
        MOV  R5,*R4         SAVE VALUE FOR LATER
        INCT R4             END LINE STORED AFTER START LINE
        INC  R1             GET READY TO GET NEXT VALUE FROM XB
        CI   R1,4           BOTH START AND END LINE STORED?
        JLT  GLINE          NO, GET END LINE
        CLR  R0
        BLWP @NUMREF        YES, GET TAB VALUE
        CLR  R5
        MOV  @FAC,R4        MOVE VALUE FROM FAC TO R4
        ANDI R4,>00FF       VALUE IS IN LOWER BYTE
LOOP3   INC  R5             START BINARY TO BCD CONVERSION
        AI   R4,-10                   R5 COUNTS "TENS"
        JLT  C3                       R4 COUNTS "ONES"
        JMP  LOOP3
C3      DEC  R5
        AI   R4,10
        SWPB R5
        MOVB R5,R4          STORE "TENS" AS HIGH BYTE OF "ONES"
        AI   R4,>3030       CONVERT TO ASCII
        MOV  R4,@TAB        STORE IN TAB PORTION OF FIRST TL.
*
        CLR  R0
        LI   R1,1           NOW WE WANT THE FIRST VALUE FROM XB
        LI   R2,FILE        STORE IT AS PART OF THE PAB
        BLWP @STRREF        GET THE STRING NOW
        LI   R0,>1E00       VDP BUFFER FOR PAB
        LI   R1,PAB
        LI   R2,>0028
        BLWP @VMBW          MOVE IT TO VDP FROM CPU
        LI   R6,>1E09
        MOV  R6,@>8356
        BLWP @DSRLNK        NOW OPEN THE DISK FILE
        DATA 8
        LI   R0,>1E00
        LI   R1,>0300
        BLWP @VSBW          MOVE WRITE BYTE TO PAB
*
        MOV  R14,R14        SINGLE DENSITY DUMP?
        JEQ  SD             YES, DON'T CHANGE ANYTHING
        INC  @DENS          NO, CHANGE DENSITY AND
        INC  @LEN               PRINT LINE LENGTH IN FIRST TL.
SD      LI   R0,>1E05
        LI   R1,>2B00       LENGTH OF FIRST TL
        BLWP @VSBW          MOVE IT TO PAB
***********************************************************
*       FIRST TL CONTAINS CODES TO INITIALIZE GRAPHICS    *
***********************************************************
        LI   R0,>1F00       DATA BUFFER IN VDP
        LI   R1,TL1
        LI   R2,>2B
        BLWP @VMBW          MOVE FIRST TL TO VDP
        MOV  R6,@>8356
        BLWP @DSRLNK        SEND IT TO THE PRINTER
        DATA 8
***********************************************************
*       EACH REDEFINABLE XB CHARACTERS PATTERN WILL BE    *
*       STORED AS A TRANSLITERATE                         *
***********************************************************
        LI   R10,1024       POINT TO START OF IMAGE TABLE
L0      MOV  R10,R0
        LI   R1,IN          WE'LL STORE THE PATTERN HERE
        LI   R2,8
        BLWP @VMBR          GET A PATTERN
        LI   R5,128         R5 POINTS TO BIT BEING CONVERTED
        CLR  R8             R8 POINTS TO BYTE IN CONVERTED PATTERN
L3      LI   R9,128         R9 POINTS TO BYTE NUMBER
        CLR  R3             R3 POINTS TO BYTE BEING CONVERTED
        CLR  R4             R4 HOLDS CONVERTED BYTE
L2      CLR  R7             R7 HOLDS BYTE BEING CONVERTED
***********************************************************
*       CONVERT PATTERN                                   *
***********************************************************
        MOVB @IN(3),R7
        SWPB R7
        C    R7,R5
        JLT  L1
        A    R9,R4
```

-->

```
        S     R5,R7
        SWPB  R7
        MOVB  R7,@IN(3)
L1      INC   R3
        SRA   R9,1
        JGT   L2
        SWPB  R4
        MOVB  R4,@DO(8)
        INC   R8
        SRA   R5,1
        CI    R8,8
        JLT   L3
*****************************************************************
*       CHANGE TO ASCII VALUES AND STORE IN OUTPUT BUFFER       *
*****************************************************************
        CLR   R9             POINTS TO BYTE IN CONVERTED PATTERN
        CLR   R8             OFFSET FOR OUTPUT BUFFER
*       ANOTHER BINARY TO BCD CONVERSION               *
LDTL    CLR   R4             R4 COUNTS "ONES"
        CLR   R5             R5 COUNTS "TENS"
        CLR   R7             R7 COUNTS "HUNDREDS"
        MOVB  @DO(9),R4
        SWPB  R4
LOOP    INC   R5
        AI    R4,-10
        JLT   C1
        JMP   LOOP
C1      DEC   R5
        AI    R4,10
        CI    R5,10
        JLT   L100
LOOP2   INC   R7
        AI    R5,-10
        JLT   C2
        JMP   LOOP2
C2      DEC   R7
        AI    R5,10
*       DON'T PRINT ANY LEADING ZEROS HERE        *
L100    MOV   R7,R7
        JEQ   ZERO1
        MOVB  @ASCII(7),@TLDATA(8)
        INC   R8
ZERO1   MOV   R5,R5
        JEQ   ZERO2
        MOVB  @ASCII(5),@TLDATA(8)
        INC   R8
ZERO2   MOVB  @ASCII(4),@TLDATA(8)
        INC   R8
        MOVB  @COMMA,@TLDATA(8)
        INC   R8
        MOV   R14,R14        SINGLE DENSITY?
        JEQ   SD6
        MOV   R7,R7          IF NOT, REPEAT LAST CHARACTER IN BUFFER
        JEQ   ZERO3
        MOVB  @ASCII(7),@TLDATA(8)
        INC   R8
ZERO3   MOV   R5,R5
        JEQ   ZERO4
        MOVB  @ASCII(5),@TLDATA(8)
        INC   R8
ZERO4   MOVB  @ASCII(4),@TLDATA(8)
        INC   R8
        MOVB  @COMMA,@TLDATA(8)
        INC   R8
SD6     INC   R9
        CI    R9,8           LAST BYTE?
        JLT   LDTL           IF NOT, GET NEXT
*****************************************************************
*       OUTPUT TRANSLITERATE                                    *
*****************************************************************
        AI    R8,7           COMPUTE TOTAL LINE LENGTH
        BL    @NXT           GET NEXT ASCII TRANSLITERATE VALUE
        LI    R0,>1E05
        MOV   R8,R1
        SWPB  R1
        BLWP  @VSBW          WRITE LINE LENGTH TO PAB
        LI    R0,>1F00
```

-->

```
       LI    R1,TLBUF
       MOV   R8,R2
SD4    BLWP  @VMBW          PUT LINE IN VDP
       MOV   R6,@>8356
       BLWP  @DSRLNK        NOW OUTPUT IT TO DISK
       DATA  8
       AI    R10,8          POINT TO NEXT IMAGE
       CI    R10,1903       LAST ONE?
       JGT   SCDMP
       B     @LO            IF NOT, DO NEXT ONE
**************************************************************
*      DUMP IMAGE TO DISK FILE                               *
**************************************************************
SCDMP  LI    R0,>1E05
       LI    R1,>2100
       BLWP  @VSBW          PUT LENGTH OF IMAGE LINE IN PAB
       MOV   @STARTL,R5     GET STARTING LOCATION AND
       MOV   @ENDL,R7       ENDING LOCATION
       INC   R7
LOOPB  CLR   R4
LOOPC  MOV   R5,R0
       BLWP  @VSBR          READ CHARACTER FROM IMAGE TABLE
       SRL   R1,8           MOVE TO LOWER ORDER BYTE
       AI    R1,-96         ADJUST FOR BASIC
       CI    R1,32          LESS THAN LEGAL GRAPHIC CHAR?
       JGT   CONT1
       LI    R1,32          IF SO, DEFAULT TO CHR$(32)
CONT1  CI    R1,143         GREATER THAN LEGAL?
       JLT   CONT2
       LI    R1,143         IF SO, DEFAULT TO CHR$(143)
CONT2  AI    R1,-32         ADJUST R1 TO BECOME OFFSET FOR "SCREEN" DATA
       MOVB  @SCREEN(1),@BUFDTA(4)
       INC   R4
       INC   R5
       CI    R4,32          END OF LINE?
       JLT   LOOPC          IF NOT, GET NEXT IMAGE
       LI    R0,>1F00
       LI    R1,BUFFER
       INC   R4
       MOV   R4,R2
       BLWP  @VMBW          IF SO, MOVE LINE TO VDP
       MOV   R6,@>8356
       BLWP  @DSRLNK        THEN OUTPUT TO DISK
       DATA  8
       C     R5,R7          LAST LINE?
       JLT   LOOPB          IF NOT, DO NEXT
**************************************************************
*      RESET TRANSLITERATE CODES                             *
**************************************************************
       LI    R0,>1E05
       LI    R1,>0B00
       BLWP  @VSBW          CHANGE LINE LENGTH IN PAB
       LI    R4,>3030       \
       MOV   R4,@DEC3        \    RESET TRANSLITERATE BUFFER
       AI    R4,>0100        /    TO .TL 001
       MOVB  R4,@DEC1       /
RST    MOVB  @DEC3,@TLDATA          TRANSLITERATE THE
       MOVB  @DEC2,@TLDATA+1             VALUE
       MOVB  @DEC1,@TLDATA+2           TO ITSELF
       LI    R0,>1F00
       LI    R1,TLBUF
       LI    R2,>000B
       BLWP  @VMBW          PUT IT IN VDP
       MOV   R6,@>8356
       BLWP  @DSRLNK        OUTPUT IT TO THE DISK
       DATA  8
       MOV   @DEC3,R5       \
       CI    R5,>3132        \
       JLT   L12              \      HAVE ALL VALUES
       MOVB  @DEC1,R5         /        BEEN RESET?
       SRL   R5,8            /
       CI    R5,>32         /
       JEQ   EXIT           IF YES, GET READY TO RETURN
L12    BL    @NXT           IF NOT, CALCULATE NEXT TL VALUE
       JMP   RST
**************************************************************
*      CLOSE DISK FILE AND RETURN TO XB                      *
**************************************************************
```

-->

```
EXIT    LI   R0,>1E00
        LI   R1,>0100
        BLWP @VSBW        PUT CLOSE BYTE IN PAB
        MOV  R6,@>8356
        BLWP @DSRLNK      CLOSE FILE
        DATA 8
        LWPI >83E0        RESET WS POINTER
        MOV  @SAVE,R11    GET RETURN VALUE
        B    *R11         RETURN TO XB
*************************************************************
*       ROUTINE TO INCREMENT ASCII TL VALUE                 *
*************************************************************
NXT     CLR  R4
        MOVB @DEC1,R4     MOVE "ONES" BYTE TO R4
        AI   R4,>0100     INCREMENT IT AND MOVE
        MOVB R4,@DEC1        IT BACK
        CI   R4,>3A00     IS IT GREATER THAN ASCII 9 (CHR$(57))?
        JLT  L10
        LI   R4,>3000
        MOVB R4,@DEC1     IF SO, REPLACE THE VALUE WITH ASCII 0
        MOVB @DEC2,R4          AND INCREMENT
        AI   R4,>0100           THE "TENS"
        MOVB R4,@DEC2             VALUE
        CI   R4,>3A00     IS THE "TENS" VALUE GREATER THAN ASCII 9?
        JLT  L10
        LI   R4,>3000
        MOVB R4,@DEC2     IF SO, REPLACE THE VALUE WITH ASCII 0
        MOVB @DEC3,R4          AND INCREMENT THE
        AI   R4,>0100              "HUNDREDS"
        MOVB R4,@DEC3                 VALUE
**
**      CHECK IF THE VALUE IS ONE THAT WE DON'T
**      WANT TO TRANSLITERATE
**
L10     MOVB @DEC1,R9
        SWPB R9
        MOVB @DEC2,R9
        CI   R9,>3130
        JEQ  NXT
        CI   R9,>3133
        JEQ  NXT
        CI   R9,>3237
        JEQ  NXT
        CI   R9,>3332
        JEQ  NXT
        CI   R9,>3338
        JEQ  NXT
        CI   R9,>3432
        JEQ  NXT
        CI   R9,>3436
        JEQ  NXT
        CI   R9,>3634
        JEQ  NXT
        CI   R9,>3934
        JEQ  NXT
        RT                RETURN WHEN OK
**********************************************************************
*                                                                    *
* NOTE:  SINCE THE EXTENDED BASIC LOADER DOES NOT RECOGNIZE THE DSRLNK *
*          UTILITY, IT WAS NECESSARY TO INCLUDE IT.                  *
*                                                                    *
**********************************************************************
*
* BEGINNING OF DSRLNK ROUTINE
*
DSRLNK DATA DSRREG,DSR0
DSR0    MOV  *14+,5
        SZCB @DATA2,15
        MOV  @>8356,0
        MOV  0,9
        AI   9,>FFF8
        BLWP @VSBR
        MOVB 1,3
        SRL  3,8
        SETO 4
        LI   2,NAME
DSR2    INC  0
        INC  4
```

-->

```
         C     4,3
         JEQ   DSR1
         BLWP  @VSBR
         MOVB  1,*2+
         CB    1,@DATA3
         JNE   DSR2
DSR1     MOV   4,4
         JEQ   DSR3
         CI    4,7
         JGT   DSR3
         CLR   @>83D0
         MOV   4,@>8354
         MOV   4,@BUFF3
         INC   4
         A     4,@>8356
         MOV   @>8356,@BUFF4
         LWPI  >83E0
         CLR   1
         LI    12,>0F00
DSR6     MOV   12,12
         JEQ   DSR4
         SBZ   0
DSR4     AI    12,>0100
         CLR   @>83D0
         CI    12,>2000
         JEQ   DSR5
         MOV   12,@>83D0
         SBO   0
         LI    2,>4000
         CB    *R2,@DATA1
         JNE   DSR6
         A     @DSRREG+10,2
         JMP   DSR7
DSR9     MOV   @>83D2,2
         SBO   0
DSR7     MOV   *2,2
         JEQ   DSR6
         MOV   2,@>83D2
         INCT  2
         MOV   *2+,9
         MOVB  @>8355,5
         JEQ   DSR8
         CB    5,*2+
         JNE   DSR9
         SRL   5,8
         LI    6,NAME
DSR10    CB    *6+,*2+
         JNE   DSR9
         DEC   5
         JNE   DSR10
DSR8     INC   1
         MOV   1,@BUFF5
         MOV   9,@BUFF2
         MOV   12,@BUFF1
         BL    *9
         JMP   DSR9
         SBZ   0
         LWPI  DSRREG
         MOV   9,0
         BLWP  @VSBR
         SRL   1,13
         JNE   DSR11
         RTWP
DSR5     LWPI  DSRREG
DSR3     CLR   1
DSR11    SWPB  1
         MOVB  1,*13
         SOCB  @DATA2,15
         RTWP
*
NAME     BSS   14                  NAME BUFFER
DSRREG   BSS   32                  WORKSPACE FOR DSRLNK
DATA1    DATA  >AA00
DATA2    DATA  >2000
DATA3    DATA  >2E00
BUFF0    BSS   2
BUFF1    BSS   2
BUFF2    BSS   2
```


-->

```
BUFF3   BSS  2
BUFF4   BSS  2
BUFF5   BSS  2
*
* END OF DSRLNK ROUTINE
*
MYREGS BSS  32
SAVE    DATA >0000
ASCII   DATA >3031,>3233,>3435,>3637,>3839
COMMA   DATA >2C00
PAB     DATA >0012,>1F00,>5000,>0000
        BYTE >00
FILE    BYTE >1F
        BSS  >1F
        EVEN
TL1     TEXT '.TL 1:27,65,8,10,13,27,68,'
TAB     TEXT '18'
        TEXT ',0,9,27,'
DENS    TEXT '75'
        TEXT ',0,'
LEN     TEXT '1'
CR      BYTE >0D
        EVEN
IN      BSS  8
DO      BSS  8
TLBUF   TEXT '.TL '
DEC3    BYTE >30
DEC2    BYTE >30
DEC1    BYTE >31
        BYTE >3A
TLDATA BSS  72
        EVEN
BUFFER BYTE >01
BUFDTA BSS  32
        EVEN
STARTL DATA >0000
ENDL    DATA >0300
SCREEN DATA >0203,>0405,>0607,>0809
        DATA >0B0C,>0E0F,>1011,>1213,>1415
        DATA >1617,>1819,>1A1C,>1D1E,>1F21
        DATA >2223,>2425,>2728,>292B,>2C2D
        DATA >2F30,>3132,>3334,>3536,>3738
        DATA >393A,>3B3C,>3D3E,>3F41,>4243
        DATA >4445,>4647,>4849,>4A4B,>4C4D
        DATA >4E4F,>5051,>5253,>5455,>5657
        DATA >5859,>5A5B,>5C5D,>5F60,>6162
        DATA >6364,>6566,>6768,>696A,>6B6C
        DATA >6D6E,>6F70,>7172,>7374,>7576
        DATA >7778,>797A
        END
```

THIS IS A TABLE OF ALL THE CHARACTERS (IN HEX) THAT WE WILL TRANSLITERATE

---

99 POTPOURRI

News, Corrections, Updates, Editorials, Kudos and Come-what-may

I WISH I HAD:

Fulfillments:

F2: For John Singleton, Westlake, LA. MENGEN, available on the TI FORUM on CIS, converts an Extended BASIC screen to Assembly object code for linking to your program. Graphics are supported, except character 130. A few screens can be loaded at once and using CALL INIT will allow loading another set of screens (your RAM Disk will help!).

Wishes:

W3: A program to dump graphics and text to my Pro-Writer #8510 printer. I'd like to press a <CTRL> or <FCTN> key for the dump. F.J. Bubenik, Jr., Hicksville, NY.

The former manager of NCC has now formed her own discount disk firm. Contact Renee' Dezarn, 87 Rhoades Court, San Jose, CA 95126 today!

COMING SOON:

Surprises! New products from Bytemaster and more new staff members for *Super 99 Monthly*!

```
##########################################################################
# NEXT MONTH:  Warren Agee's second FORTH tutorial                       #
#              Navarone DBM tips                                         #
#              TI-Artist II tutorial                                     #
#              Extended BASIC tips                     And Much More!!!  #
##########################################################################
```

-->

SUBSCRIPTIONS (PER YEAR):

U.S. AND POSSESSIONS	
FIRST CLASS	$16.00
THIRD CLASS	$12.00
OTHER COUNTRIES	
AIR MAIL	$26.50
SURFACE MAIL	$16.00
INDIVIDUAL COPIES:	
U.S. SUBSCRIBERS	
FIRST CLASS	$ 1.35
THIRD CLASS	$ 1.00
CANADA SUBSCRIBERS	$ 1.35
OTHER	$ 1.50

Check or Money Order in U.S. funds, coded for processing through the U.S. Federal Reserve Bank System. No billings or credit sales. (all issues available at press time)

SUPER 99 MONTHLY ORDER FORM

NAME________________________________

ADDRESS_____________________________

CITY___________________STATE_______

ZIP_________COUNTRY_________________

For back issues, specify which:

____

READER FEEDBACK: (Attach comments)

SUPER 99 MONTHLY is published monthly by Bytemaster Computer Services, 171 Mustang Street, Sulphur, LA 70663. All correspondence received will be considered unconditionally assigned for publication and copyright and subject to editing and comments by the editors of *SUPER 99 MONTHLY*.

STANDARD KEY

1	Computer	A	TI-99/4A
2	Module	XB	Extended BASIC
		TW	TI-Writer
		EA	Editor/Assembler
3	RS-232	B	TI
4	Disk Drive	B	TEAC 55B
5	Expansion Box	A	TI
6	Disk Controller	B	CorComp
7	Memory Card	B	MYARC MEXP-1 (128K)
9	Monitor or TV	B	TI Color Monitor
10	Printer	B	Gemini 15-X

GRAM Kracker and DisKassembler are registered trademarks of Millers Graphics.

EDITOR

Richard M. Mitchell (CIS 70337,1011)

CORRESPONDING STAFF WRITERS

Barry A. Traver
Charles M. Robertson
Steven J. Szymkiewicz, MD

Bytemaster Computer Services
171 Mustang Street
Sulphur, LA 70663

Bulk Rate
U.S. Postage
PAID
Sulphur, LA 70663
Permit No. 141

POSTMASTER: ADDRESS CORRECTION REQUESTED.
RUSH -- TIME DATED MATERIAL.